社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2007－DBS－143（46）
2007／7／3

XML情報検索システムの検索結果に対する入れ子を考慮した順序付け

清水 敏之† 吉川 正俊†

† 京都大学 情報学研究科 〒606-8501 京都市左京区吉田本町
E-mail: †shimizu@soc.i.kyoto-u.ac.jp, ††yoshikawa@i.kyoto-u.ac.jp

あらまし XML情報検索ではXML文書中の部分文書を対象にして検索を行い，検索結果となる部分文書には入れ子が存在する場合がある．検索結果となる部分文書に入れ子がある場合，検索結果を閲覧していくと，既に見たことのある内容が再び出現することがある．我々はXML情報検索システムの検索結果に対して，結果から得られる利得と結果を閲覧するためのコスト考え，効率的に問合せに対する情報を取得するための検索結果の順序付けを考えた．
キーワード XML，情報検索，順序付け

On Ranking Method for Search Results of XML Information Retrieval System Considering Nesting

Toshiyuki SHIMIZU† and Masatoshi YOSHIKAWA†

† Graduate School of Informatics, Kyoto University Yoshida-Honmachi, Sakyo-ku, Kyoto, 606-8501 Japan
E-mail: †shimizu@soc.i.kyoto-u.ac.jp, ††yoshikawa@i.kyoto-u.ac.jp

Abstract XML information retrieval systems search for relevant document fragments in XML documents for input queries. In general, elements are considered to be search units, and therefore search results may have nesting elements. If a system order result elements simply by their relevances, we may browse the same content more than once due to the nestings. We propose the ranking method that enables us to browse search results of XML information retrieval systems efficiently by introducing the concepts of *benefit* and *reading effort*.
Key words XML, Information Retrieval, Ranking Method

1. はじめに

大量のXML文書の中から，ある話題に関して知りたい時，キーワードを用いて検索を行い，関連する部分のみを取得することが考えられる．XML情報検索では，一般にXML文書中の要素を検索の単位とし，問合せに関連する要素を順序付けして利用者に提示する．例えば，XML化された論文の場合，問合せに関連する節や段落を関連度に応じて順序付けして取得する．

INEX 2005 [1] ではXML情報検索システムの精度評価のために，要素を取得する三つの戦略を定義している．*Thorough* では単純に全ての要素を関連度順に取得する．*Thorough* では検索結果の要素中に入れ子により内容に重なりがある場合がある．*Thorough* での検索結果中の先祖子孫関係にある要素から最も関連度の高いもの (*focussed node*) を取り出すことで重複を排除した検索結果を取得するのが *Focussed* である．*Focussed* では冗長性を排除できるが，取得された要素以外の要素の関連度は分からず，XML情報検索システムの利点を損なう可能性がある [2]．そして，関連する文脈の要素を連続して取得するために，同一の文書内の要素をまとめて取得するのが *FetchBrowse* である．

我々はXML情報検索における検索結果の提示に関して，以下のような問題点があると考えた．

- 入れ子への対処

実際に利用者が入れ子になっている要素の内容を閲覧する際には，入れ子の親の方の要素を内容を閲覧することで，その要素に含まれる全ての子要素の内容は同時に閲覧することが可能となる．そのため，検索結果の提示の際には入れ子を考慮することが重要であるが，INEX 2005 の検索戦略のうち，唯一入れ子を考えている *Focussed* では *focussed node* 以外のノードを取得しないため，柔軟性に欠ける．

- 要素サイズの多様性

また，XML情報検索では検索結果として返ってくる要素の大きさは多様性に富み，大きな要素，例えば根要素 (文書全体) を返すこともあれば，小さな要素を返すこともある．そのため，検索結果を上位からk件取得するtop-k検索によって得られる要素集合に関し，どれくらい読む量があるのか不明である．

以上の問題点に対処するため，我々は，検索結果から得られる *benefit* と検索結果を閲覧するのに必要な *reading effort* に

基づく新たな評価指標を提案する．利用者は検索結果に対して支払う *reading effort* の量を指定し，システムはその *reading effort* 内で得られる *benefit* を最大化するように結果を提示する．検索結果に対するスコアは，その結果から *benefit* を取得する効率性であると見ることができ，*benefit* を *reading effort* で割ることにより求まるものとした．

XML 情報検索の検索結果中の入れ子の扱いに関する研究としては，文献 [2] があげられる．文献 [2] では，ある要素が検索結果として出力されると，その子孫要素と先祖要素のスコアを再計算し，それ以降の結果を再ランキングする．本研究では，利用者が検索結果に対して支払う *reading effort* の量を指定することを想定し，*benefit* と *reading effort* を用いて検索結果の順序付けを行う．さらに，その評価に関して，INEX 2005 における XML 情報検索のテストコレクションにおける正解データを利用した手法を考えた．

2. Benefit と Reading Effort

検索結果要素に包含関係がある場合，検索結果を閲覧していくと，既に見たことのある内容が出現することがある．我々は検索結果に対して *benefit* と *reading effort* を導入することで効率的な検索結果の提示ができると考えた．本節では，*benefit* と *reading effort* に関し，その特徴を議論する．

2.1 Benefit

benefit は問合せに関して得られる利得である．基本的には，ある要素の *benefit* はその子要素の *benefit* の和と考えられるが，まとめて読むことにより内容を補完しているといった状況も考えられるため，和よりも多少大きな *benefit* を取得できることも考えられる．*benefit* に関し，以下のような特徴があると考えた．具体的な *benefit* 値の算出に関しては後で議論する．

特徴 1 それぞれのノードの *benefit* はその子ノードの *benefit* の和と等しいか，和よりも大きくなる．

要素 e の *benefit* を $e.benefit$ とする．

2.2 Reading Effort

reading effort は読むのにかかるコストである．算出にはテキスト長を利用するのが適切であると思われる．基本的には，ある要素の *reading effort* はその子要素の *reading effort* の和と考えられるが，まとめて読むことによりそれぞれ個別に読むよりも楽に読むことが可能であると思われるため，和よりも多少小さな *reading effort* となることも考えられる．*reading effort* に関し，以下のような特徴があると考えた．具体的な *reading effort* 値の算出に関しては後で議論する．

特徴 2 それぞれのノードの *reading effort* はその子ノードの *reading effort* の和と等しいか，和よりも小さくなる．

要素 e に対する *reading effort* を $e.reading\_effort$ とする．*reading effort* は問合せには依存せず，要素それ自体に応じて決まる．

3. Benefit と Reading Effort に基づく順序付け

要素に対して付与された *benefit*, *reading effort* を基に，効率的に検索結果を閲覧できるように検索結果の順序付けを行う．要素に対するスコアは *benefit* を *reading effort* で割ることにより求められるものとする．このスコアは結果から *benefit* を取得する効率性であると見ることができる．要素 e のスコアは $e.score$ とし，$e.score = e.benefit/e.reading\_effort$ である．

順序付けを考える際には入れ子による内容の重複を考慮する必要がある．重複には 2 つのパターンが考えられる．

（1） **Contain** ある結果の内容がそれより上位のランクの結果の内容を含んでいる．

（2） **Contained** ある結果の内容がそれより上位のランクの結果の内容に含まれている．

利用者が支払う *reading effort* を指定し，システムはその *reading effort* 内で *benefit* を最大化するように結果を取得する．この問題は，入れ子による制約のあるナップサック問題の一種であると考えられるが，実際の検索システムでは検索結果の連続性が重要であると思われるため，入れ子を考慮に入れつつ，貪欲にスコアの高いものから取得していくことが現実的である．ここで検索結果の連続性とは以下で定義される特徴である．

定義 1 検索結果の連続性
ある *reading effort* 値の指定に対する結果要素集合の内容は，指定する *reading effort* 値を増やしたときに得られる結果要素集合の内容に含まれる．

まず，*benefit* と *reading effort* を利用して要素に対して付与されたスコアを利用し，*Thorough* により結果要素を取得する．その後，入れ子による内容の重複を取り除きつつ，上位から利用者が指定した *reading effort* の量に至るまで要素を取得する．

入れ子による内容の重複の除去は，以下を行う．**Contain** のパターンが出現した際には，その時点で以前に出現したその結果の子孫要素を取り除き，その結果を追加する．**Contained** のパターンでは，既にその内容は取得されているため，単純にその結果をスキップする．

ある結果要素 e を取得すると，その先祖要素 e_a の *benefit* と *reading effort* は e の影響を受ける．e を取得した後に e_a を取得する場合，e_a によって得られる *benefit* の増加分は $e_a.benefit - e.benefit$ となる．また，その増加分の *benefit* を取得するためにかかる *reading effort* の増加分は $e_a.reading\_effort - e.reading\_effort$ となる．

利用者が *reading effort* の閾値を入力すると，その *reading effort* で取得可能な結果を入れ子を考慮して取得するためのアルゴリズムは図 1 のようになる．

例えば，図 2 のように *benefit* と *reading effort* が付与され，*reading effort* の閾値として 40 が設定されたとき，結果をどのように取得するか説明する．図 2 は，XML を木構造で表現し，*benefit* と *reading effort* を要素の隣に *benefit*/*reading effort* の

入れ子を考慮した順序付け.

Input: $list_{in}$, // result list of *Thorough*
$reading\_effort_t$ // threshold of *reading effort*
Output: $list_{out}$
$reading\_effort_c = 0$ // Cumulated reading effort
while $((e = top(list_{in}))! = null)$ **do**
  remove e from $list_{in}$
  $skip = false$
  $remove = empty$
  for (e_o in $list_{out}$) **do**
    if (e_o is ancestor of e) **then**
      skip = true
      break
    end if
    if (e_o is descendant of e) **then**
      add e_o to $remove$
    end if
  end for
  if ($skip$) **then**
    continue
  end if
  $reading\_effort_c += e.reading\_effort$
  if ($reading\_effort_c > reading\_effort_t$) **then**
    break
  end if
  for ($e_d \in remove$) **do**
    remove e_d from $list_{out}$
  end for
  add e to $list_{out}$
  for ($e_a \in e.ancestors$) **do**
    $e_a.benefit -= e.benefit$
    $e_a.reading\_effort -= e.reading\_effort$
    $rerank\ e_a$ in $list_{in}$
  end for
end while
return $list_{out}$

図 1 入れ子を考慮した順序付け.

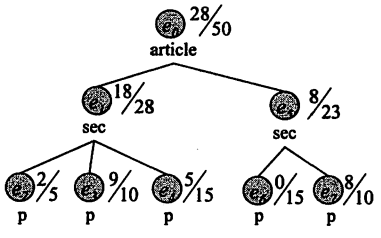

図 2 *benefit* と *reading effort* の付与例.

形式で示している. ここでは簡単化のため, 具体的な *benefit* と *reading effort* の算出式は想定していない.

まず, *Thorough* により要素を順序付けし, リストを得る. この場合, $e_3(e_3.score = 9/10 = 0.9)$, $e_7(0.8)$, $e_1(0.64)$, $e_0(0.56)$, $e_2(0.4)$, $e_5(0.35)$, $e_4(0.33)$ の順になる. このリスト $list_{in}$ を上位から順番に処理していく. まず, e_3 を処理し, 結果リスト $list_{out}$ に追加する. その際, 先祖である e_1 と e_0 には *benefit* と *reading effort* の調整を行う. この場合, e_1 に関しては, $e_1.benefit = 9$, $e_1.reading\_effort = 18$, $e_1.score = 9/18 = 0.5$ となり, e_0 に関しては, $e_0.benefit = 19$, $e_0.reading\_effort = 40$, $e_0.score = 19/40 = 0.48$ となる. この調整を反映し, $list_{in}$ 中の要素を再順序付けする. この場合, $e_7(0.8)$, $e_1(0.5)$, $e_0(0.48)$, $e_2(0.4)$, $e_5(0.35)$, $e_4(0.33)$ となる. 次に e_7 を処理し, $list_{out}$ は e_3, e_7 となり, $list_{in}$ は $e_1(0.5)$, $e_2(0.4)$, $e_0(0.37)$, $e_4(0.33)$, $e_5(0)$ となる. さらに e_1 を処理するが, この際 $list_{out}$ 中の e_3 は e_1 の子孫であるため, $list_{out}$ から取り除く. $list_{out}$ は e_7, e_1 となり, $list_{in}$ は $e_2(0.4)$, $e_4(0.33)$, $e_0(0.17)$, $e_5(0)$ となる. 続いて e_2 と e_4 を処理しようとするが, その先祖である e_1 が既に $list_{out}$ 中にあるため, 処理がスキップされ, e_0 を処理しようとするが, 指定された *reading effort* の閾値を超えてしまうため, 処理を終了し, その時点の $list_{out}$ である e_7, e_1 が結果として取得される.

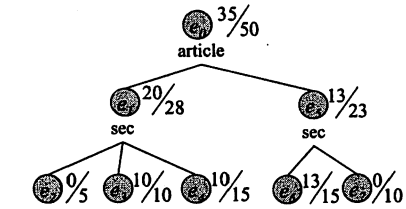

図 3 実際の *benefit* と *reading effort*.

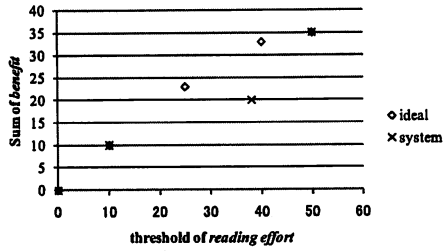

図 4 b/e グラフ.

4. システムの評価

benefit と *reading effort* を利用して 3. 節の順序付けを行ったシステムの評価には, ある *reading effort* の閾値を利用者が設定したときに, 理想的なシステムによって得られる結果中の *benefit* の総量と, 評価対象のシステムによって得られる結果中の *benefit* の総量を比較することで行うことができると考えた. ここで理想的なシステムとは, 要素に対する実際の *benefit* を利用して順序付けするシステムである. XML 情報検索システムの実装者は, より実際の *benefit* に近い *benefit* を推測することで, よりよいシステムを目指すことになる. *reading effort* に関しては, 問合せに依存しない値であり, 理想的なシステムと評価対象のシステムで共通の値が利用可能であると考えた.

例えば, システムは図 2 のように *benefit* と *reading effort* を算出したが, 実際には図 3 のように *benefit* と *reading effort* が分布していた場合は, *reading effort* の閾値 40 に対して, システムは e_7, e_1 を取得し, 合計 20 の *benefit* を獲得するが, 理想的なシステムは e_3, e_6, e_4 を取得し, 合計 33 の *benefit* を獲得する.

利用者が設定する *reading effort* の閾値を変化させ, 理想的なシステムによる *benefit* の総量と評価対象のシステムによる *benefit* の総量をプロットしたグラフを benefit/effort グラフ (以

下，b/e グラフと略す) と呼び，評価として利用する．実際の *benefit* と *reading effort* の分布は図 3 のようであり，システムが算出した *benefit* と *reading effort* が図 2 のようである場合，b/e グラフは図 4 のようになる．図 4 では，理想的なシステムを ideal とし，評価対象のシステムを system としている．

このグラフでは，ある与えられた *reading effort* 値 r に対して得られる *benefit* は，r 以下の最大の *reading effort* 値を持つ点の *benefit* となる．たとえば *reading effort* 値として 30 を与えた場合，得られる *benefit* は ideal では 23，system では 10 となる．b/e グラフを描くことにより，理想的なシステムに対し，評価対象システムがどの程度の良さなのか，直観的に把握できる．

5. 実　　験

システム評価のために，INEX プロジェクト [3] の提供する XML 情報検索システムのためのテストコレクションを利用することを考えた．このテストコレクションは XML 文書集合，問合せ集合 (Topics)，問合せに対する正解データ (Assessments) から成る．*benefit* と *reading effort* を用いる簡単なシステム実装を行い，いくつかの INEX 2005 の Topic に関し，b/e グラフを描いてみた．

5.1 INEX 2005 の正解データ

INEX 2005 では問合せに対する正解データは Exhaustivity (ex) と Specificity (sp) の二次元の情報から成る．Exhaustivity はどの程度問合せに対して議論しているかの指標であり，Highly exhaustive (HE), Partially exhaustive (PE), Not exhaustive (NE) の 3 段階を考えている(注1)．HE を 1, PE を 0.5, NE を 0 とする．Specificity は要素の内容全体に対しどの程度問合せに関連する内容になっているかの指標であり，問合せに関連する内容の長さ (rsize) を要素の内容全体の長さ (size) で割ることにより求めている．要素 e の ex, rsize, size を $ex(e)$, $rsize(e)$, $size(e)$ とすると，ex, rsize, size の性質から以下の式が成り立つ．ここで，$e.parent$ を e の親要素とし，$e.children$ を e の子要素集合とする．

$$ex(e) \leqq ex(e.parent) \tag{1}$$

$$rsize(e) = \sum_{e_i \in e.children} rsize(e_i) \tag{2}$$

$$size(e) = \sum_{e_i \in e.children} size(e_i) \tag{3}$$

なお，INEX 2006 の正解データでは Specificity のみを用いている．

5.2 INEX 2005 の正解データを用いた Benefit と Reading Effort の取得

INEX 2005 の正解データから要素に対する *benefit* と *reading effort* を取得することを考えた．**特徴 1** と**特徴 2** を満足した式を得るため，以下の式を考えた．

$$e.benefit = ex(e)^{\alpha} * rsize(e)^{\beta} \quad (\alpha \geqq 0, \beta \geqq 1) \tag{4}$$

$$e.reading\_effort = size(e)^{\gamma} \quad (0 \leqq \gamma \leqq 1) \tag{5}$$

INEX 2006 の正解データでは Specificity のみを用いているが，上記の式では $\alpha = 0$ とすることで容易に対応できる．

式 1, 2 より，式 4 は**特徴 1** を満足する．また，式 3 より，式 5 は**特徴 2** を満足する．

5.3 システム実装

reading effort に関しては，問合せに依存しない値であり，理想的なシステムと評価対象のシステムで共通の値が利用可能であると考えているため，システムの実装の焦点は *benefit* の算出にある．**特徴 1** を満足するような実際の *benefit* を推測していくことになる．

ここでは，$tf - ief$ を基に *benefit* の算出式を得ることにした．$tf - ief$ 値が大きい，つまり特定性の高い語が高頻度で現れる要素ほど検索語に関する情報が得られると思われるため，*benefit* は大きくなると思われる．また，複数の検索語を用いる場合，より多くの種類の語が出現する要素ほど *benefit* が大きくなると考えた．

$$e.benefit = ex * \sum_{t \in q} tf * ief \tag{6}$$

$$ief = ln\frac{N+1}{ef} \tag{7}$$

$$ex = \frac{n}{|q|} \tag{8}$$

ここで，q は問合せであり，$|q|$ は問合せ q 中の検索語数である．ex は問合せ中の語がどの程度要素 e に出現するかの指標であり，n は問合せ中の語のうち，e に出現するものの数である．tf は索引語出現回数であり，ief は索引語の特定性である．ief の算出には全要素数 N と索引語が出現する要素の数 ef を用いる．

XML 文書に対する語の重みを求めるためにはさまざまな提案があるが [4]〜[7]，ここでは**特徴 1** を満足することが重要であると考え，**特徴 1** を満たすような簡単な式を用いた．

5.4 b/e グラフの取得

実際の *benefit* と *reading effort* は 5.2 節で得られると考え，5.3 節のシステムの b/e グラフの取得を行った．ここで $\alpha = 0.5$, $\beta = 1$, $\gamma = 1$ とした．ここでは，INEX 2005 の問合せ (Topic) 集合から四つの問合せを選択し，b/e グラフを得た．図 5 に Topic 203 の b/e グラフを，図 6 に Topic 206 の b/e グラフを，図 7 に Topic 207 の b/e グラフを，図 8 に Topic 208 の b/e グラフを示す．

6. おわりに

我々は XML 情報検索システムの結果提示に *benefit* と *reading effort* を導入することで効率的な検索結果の提示ができると考えた．利用者が *reading effort* の閾値を入力し，検索結果に対して，入れ子を考慮し，内容の重複を省きながら取得する

(注1)：要素が非常に小さい場合に対して Too Small (TS) を導入しているが，TS は NE と等価であると考える．

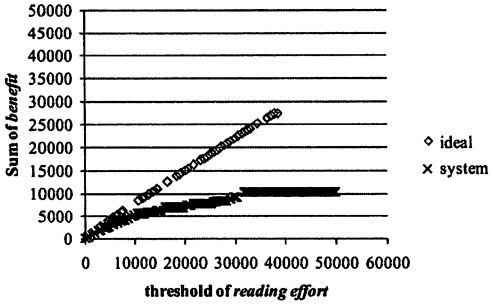

図 5 Topic 203 の b/e グラフ.

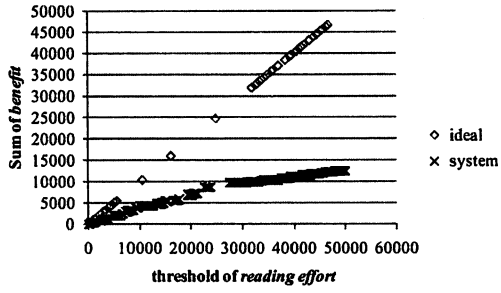

図 6 Topic 206 の b/e グラフ.

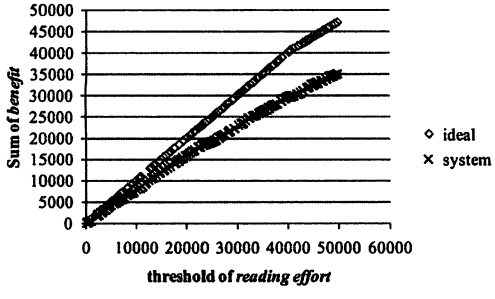

図 7 Topic 207 の b/e グラフ.

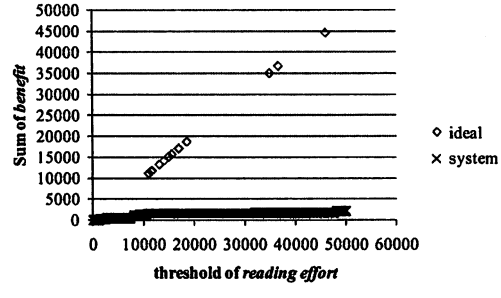

図 8 Topic 208 の b/e グラフ.

ことで効率的な検索結果の提示を実現する.

今後の課題としては，要素中で利用者に実際に読んで欲しい部分とそうでない部分を区別して表示することで，より適切な部分のみを利用者に提示することが考えられる．また，提示する結果の数が多くなると，利用者の閲覧時の煩雑さが増加すると思われる．この問題に対処するため，閲覧時の切替コスト (*switching effort*) の概念を導入することを検討している．さらに，PDF などから作成された XML 文書の場合は，検索結果を PDF のレイアウト上に重畳して提示することも考えられ[8]，そのような提示手法との統合も課題である.

文　献

[1] S. Malik, G. Kazai, M. Lalmas and N. Fuhr: "Overview of INEX 2005", In *INEX*, pp. 1–15 (2005).

[2] C. L. A. Clarke: "Controlling overlap in content-oriented XML retrieval", In *SIGIR*, pp. 314–321 (2005).

[3] INEX: "INitiative for the Evaluation of XML Retrieval". http://inex.is.informatik.uni-duisburg.de/.

[4] S. Cohen, J. Mamou, Y. Kanza and Y. Sagiv: "XSEarch: A semantic search engine for XML", In *VLDB*, pp. 45–56 (2003).

[5] T. Grabs and H.-J. Schek: "ETH Zürich at INEX: Flexible information retrieval from XML with PowerDB-XML", In *INEX*, pp. 141–148 (2002).

[6] S. Amer-Yahia, E. Curtmola and A. Deutsch: "Flexible and efficient XML search with complex full-text predicates", In *ACM SIGMOD*, pp. 575–586 (2006).

[7] T. Shimizu, N. Terada and M. Yoshikawa: "Kikori-KS: An effective and efficient keyword search system for digital libraries in XML", In *ICADL*, pp. 390–399 (2006).

[8] T. Shimizu and M. Yoshikawa: "XML information retrieval considering physical page layout of logical elements", In *WebDB* (2007).